# Optimizing Chimera Grids Using Genetic Algorithms

UCRL-JC-143856

Lars Carlsson

*Hydromechanics division,*
*Department of Naval Architecture and Ocean Engineering,*
*Chalmers University of Technology,*
*SE-412 96 Göteborg, Sweden*
lc@na.chalmers.se

N. Anders Petersson

*Center for Applied Scientific Computing,*
*Lawrence Livermore National Laboratory,*
*Livermore, CA 94551, USA*
andersp@llnl.gov

May 25, 2001

**Abstract**

We present an optimization procedure for minimizing the error in the numerical solution of a partial differential equations by modifying the computational grid. We construct exact solutions by adjusting the forcing and use the exact error in the numerical solution as objective function. Chimera (overset) grids are used to discretize the domain and a robust genetic algorithm is used to optimize the grid. Numerical examples for Poisson's equation in a domain exterior to a wing-like object demonstrate that the error in the numerical solution can be reduced significantly by using the proposed method.

# 1 Introduction

The generation of computational grids for complex geometries can be viewed as an optimization problem where the objective function is the grid quality, and the design variables are the input parameters to the grid generator. A computational grid is used to numerically solve a partial differential equation

(PDE) with a field discretization method and we call a grid optimal when it minimizes the numerical error in the solution of the PDE. However, the optimality depends on the PDE, the forcing, and the discretization method. In the current paper, we apply the "twilight-zone" forcing technique [5] to construct exact solutions of the same PDE on the same computational domain, but with a modified forcing function. Since the exact solution is known, the error can be calculated exactly and can be used as the objective function. In comparison, standard grid quality measures are based on metrics of the grid, such as cell skewness, stretching ratio between neighboring cells, or cell aspect ratio. Even though these measures are related to the truncation error in the discretization of the PDE, it is very difficult to accurately relate these measures to the actual error when the PDE is computed on the grid.

To evaluate the objective function we first need to construct a grid, compute a numerical solution to a PDE on that grid, and then evaluate the error in the numerical solution. Since it is computationally expensive to compute the numerical solution of a PDE, it is desirable to use an optimization algorithm which is able to find one very good grid in as few evaluations of the objective function as possible. Furthermore, it is hard to ensure that a grid can be constructed for all combinations of input parameters to the grid generator. And if a grid can not be constructed, the error in the numerical solution of the PDE can not be evaluated. At these points in parameter space, the objective function is defined to equal a large constant. This makes the objective function unsmooth, and the optimization algorithm must be able to handle this property.

In this report we use genetic algorithms to optimize the grids. Genetic (or evolutionary) algorithms are based on theories from natural selection [6]. Genetic algorithms are characterized by their ability to explore unknown regions of parameter space while exploiting knowledge of the regions searched so far. Other important advantages are their ability to find global optima even when the search space is unsmooth and the objective function is not differentiable. These properties make genetic algorithms more robust than traditional methods, such as steepest decent, which, for example, would run into difficulties at points in parameter space where the grid can not be constructed.

Genetic algorithms have apparently not been used much for grid optimization. Bykat [2] reported a genetic algorithm for optimizing unstructured grids. However, in his work the grid quality was estimated solely from metric properties, so it is rather different from ours.

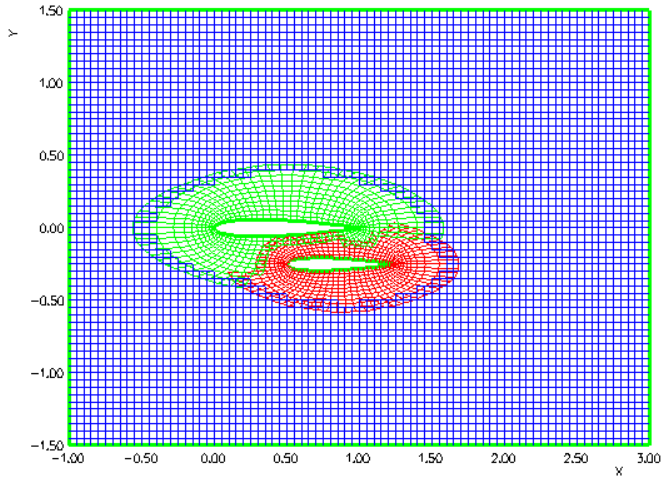


Figure 1: A region with two airfoils discretized by the Chimera technique. The green and red grids are body fitted and resolve the regions near the two airfoils. The remainder of the computational domain is discretized on a Cartesian background grid.

In the present work, we use Chimera grids [10] to handle complex geometries. In this technique, complicated computational domains are decomposed into overlapping sub-domains where each sub-domain is discretized on a structured grid. An example of a Chimera grid is shown in figure 1.

In § 2, we give an overview of the grid generator and describe the parameters governing the grid generation. Section 3 contains a specification of the optimization problem for Poisson's equation with Dirichlet conditions and in section 4, we give an overview of genetic algorithms. Finally, in § 5, we present results showing that the error in a numerical solution can be reduced significantly by optimizing the grid using the proposed method.

# 2 Generating Chimera grids

In the Chimera (or overlapping) grid technique [10], complicated computational domains are decomposed into overlapping sub-domains where each sub-domain is discretized on a structured grid. For example, in the grid shown in Figure 1, the regions near the wings are discretized on body fit-

ted curvilinear grids while the remainder of the domain is discretized on a Cartesian background grid, in which the body fitted grids can cut holes. The solution at the outer boundaries of the wing grids is connected to the solution on the Cartesian background grid through interpolation relations. Similarly, the solution at the inner, stair-stepped, boundary of the Cartesian grid is interpolated from the overlapping wing-grid.

For the generation of Chimera grids, we use the program `ogen` which is included in the Overture framework [1]. In `ogen`, there are several different component grid generation algorithms. We will focus on the Cartesian and hyperbolic grid generators, since they are the most important building blocks for making grids around general geometries. Cartesian grids are most often used to generate background grids, whereas hyperbolic grids are used to make body fitted grids near curved boundaries.

While it is trivial to generate a Cartesian grid, it is more involved to generate a hyperbolic grid. We view the grid as a mapping $\boldsymbol{x} = \boldsymbol{x}(\boldsymbol{r})$ from a unit square in parameter space $\boldsymbol{r} = (r, s)^T$ to physical space $\boldsymbol{x} = (x, y)^T$. If the grid has $N_r \times N_s$ grid points, the points in parameter space are simply $r_i = (i-1)/(N_r - 1)$, $i = 1, 2, \ldots, N_r$ and $s_j = (j-1)/(N_s - 1)$, $j = 1, 2, \ldots, N_s$. The grid points in physical space are given by $\boldsymbol{x}_{i,j} = \boldsymbol{x}(r_i, s_j)$.

The hyperbolic grid starts from a curve $\boldsymbol{x}_0(r)$ and this is taken to be the first grid line in the $s$-direction:

$$\boldsymbol{x}(r_i, s_1) = \boldsymbol{x}_0(r_i), \quad i = 1, 2, \ldots, N_r.$$

From this curve, the grid is grown in an advancing front fashion; we will call the most recently generated grid line the front. The hyperbolic grid generator in `ogen` implements a combination of the techniques developed by Chan and Steger [4], Chan and Buning [3], and Sethian [9]. In essence, the grid is generated by solving the hyperbolic-parabolic PDE:

$$\boldsymbol{x}_s = S(r, s)\boldsymbol{n} + \nu(\Delta r)^2 \boldsymbol{x}_{rr}, \tag{1}$$

Here, $\boldsymbol{n}$ is the unit normal of the front, $S(r, s)$ is the speed function, $\nu$ is the dissipation coefficient, and $\Delta r = 1/(N_r - 1)$ is the step size in the $r$-direction.

The speed function controls the cell size according to

$$\begin{aligned} S(r_i, s_j) &= d\,\frac{\overline{\Delta a_{i,j}}}{\Delta a_{i,j}}(1-\epsilon\kappa_{i,j})\alpha^{j-1}\frac{\alpha-1}{\alpha^{N_s-1}-1}, \\ \kappa_{i,j} &= \boldsymbol{x}_{rr}(r_i,s_j)\cdot\boldsymbol{n}(r_i,s_j)/|\boldsymbol{x}_r(r_i,s_j)|, \\ \boldsymbol{n}(r_i,s_j) &= (-y_r(r_i,s_j),x_r(r_i,s_j))^T/|\boldsymbol{x}_r(r_i,s_j)| \\ \Delta a_{i,j} &= |\boldsymbol{x}_r(r_i,s_j)|. \end{aligned} \tag{2}$$

The function $\overline{\Delta a_{i,j}}$ is obtained by smoothing $\Delta a_{i,j}$ by applying a Jacobi scheme $N_{smooth}$ times. The idea behind the speed function is to grow the front faster where the grid size is smaller than the (local) average grid size, and where the front is concave. The stretching of the grid in the $s$-direction is controlled by the geometric stretching factor $\alpha$. Since the step size is adjusted for the local step size and for curvature, $d$ will only approximately equal the thickness of the grid in the $s$-direction. To avoid stability restrictions on the step size in $s$, (1) is integrated implicitly with a mix of the Euler-Backwards and Crank-Nicholson schemes. The implicit mixing coefficient $I_m$, $0.5 \le I_m \le 1$, is a parameter that can be set by the user where $I_m = 1$ gives Euler-Backwards and $I_m = 0.5$ corresponds to Crank-Nicholson. After the PDE has been advanced one step in $s$, the preliminary grid line $\tilde{\boldsymbol{x}}(r, s_{j+1})$ is modified to redistribute the grid points in the $r$-direction according to the arclength and curvature along the grid line. The idea is to put more grid points where the arclength varies quickly and the curvature is large. A new parameter $\xi(r)$ is computed by equidistributing a weight function $w(r)$:

$$\begin{aligned} r &= \frac{1}{C}\int_0^{\xi(r)} w(r)\,dr, \quad C = \int_0^1 w(r)\,dr, \\ w(r) &= \gamma_s|\boldsymbol{x}_r(r)| + \gamma_\kappa\kappa(r), \end{aligned}$$

where $\gamma_s$ and $\gamma_\kappa$ are user defined. The reparametrized grid line $\boldsymbol{x}^E(r) = \tilde{\boldsymbol{x}}(\xi(r), s_{j+1})$ is then used to compute the final grid line:

$$\boldsymbol{x}(r, s_{j+1}) = (1-\gamma_e)\tilde{\boldsymbol{x}}(r, s_{j+1}) + \gamma_e\boldsymbol{x}^E(r). \tag{3}$$

The coefficient $\gamma_e$ is called the equidistribution weight, and is specified by the user. The procedure can now be repeated to compute the remaining grid lines.

In summary, the user can change the following 11 parameters in the hyperbolic grid generator: the number of grid points $N_r$ and $N_s$, the number

of smoothing steps $N_{smooth}$ for computing $\overline{\Delta a}$, the geometric growth ratio $\alpha$, the distance to march $d$, the curvature speed coefficient $\epsilon$, the dissipation coefficient $\nu$, the implicit mixing coefficient $I_m$, and the redistribution coefficients $\gamma_s$, $\gamma_\kappa$, and $\gamma_e$. Even though all parameters have reasonable default values, it is clearly a non-trivial task to choose the parameter values that yield the best grid.

# 3 The Optimization problem

In this work, we aim to minimize the error in the solution of a two-dimensional Poisson equation with Dirichlet boundary conditions. Let $\Omega \subset R^2$ be a domain with a sufficiently smooth boundary $\partial\Omega$, and let $f$ and $g$ be sufficiently smooth functions, where $f$ is defined in $\Omega$ and $g$ on $\partial\Omega$. We consider

$$\Delta\phi = f, \text{ in } \Omega, \tag{4}$$

$$\phi = g, \text{ on } \partial\Omega. \tag{5}$$

For general domains $\Omega$ and general forcing functions $f$ and $g$, it is not possible to find the exact solution $\phi$, which is necessary to compute the error in a numerical solution. To circumvent this difficulty, we will apply the "twilight-zone" forcing method [5] to construct analytical solutions for particular choices of $f$ and $g$, without making restrictions on the domain $\Omega$. In this technique, a sufficiently differentiable function $\Phi(x, y)$ is first selected. The forcing functions in (4) and (5) are then chosen to be

$$f =: \Delta\Phi, \text{ in } \Omega, \tag{6}$$

$$g =: \Phi, \text{ on } \partial\Omega. \tag{7}$$

By construction, the exact solution of (4) and (5) is $\phi(x, y) = \Phi(x, y)$. Hence, it is possible to exactly evaluate the error in the numerical solution at all grid points. Note that the same procedure applies to general PDEs, see [5] for details.

Throughout this paper, we will use the function

$$\Phi(x, y) = \cos(\pi x)\cos(\pi y)$$

as exact solution. The Poisson problems are solved numerically using the second order accurate centered finite difference technique available in the Overture [1] library.

As was described in the previous section, the construction of a Chimera grid covering $\Omega$ is governed by various parameters that we collect in a parameter vector $\boldsymbol{P} = (p_1, p_2, \ldots, p_M)^T$. We will call the corresponding grid $G(\boldsymbol{P})$. Furthermore, let the numerical approximation of $\phi$ on the grid $G(\boldsymbol{P})$ be $\phi_P$. Note that for most geometries there will be some parameter vectors $\boldsymbol{P}$ where the grid generation algorithm fails to generate a valid Chimera grid. Formally, this makes the error in the numerical solution $\phi_P$ undefined, but for practical purposes, we define it to be a large constant $e_{max} \gg 1$. Hence, we define the error in $\phi_P$ according to

$$e(\boldsymbol{P}) = \begin{cases} \|\Phi - \phi_P\|_\infty, & \text{if G}(\boldsymbol{P}) \text{ is valid,} \\ e_{max}, & \text{otherwise.} \end{cases} \tag{8}$$

The optimization problem can now be stated: Find the parameter vector $\boldsymbol{P}$ corresponding to a Chimera grid $G(\boldsymbol{P})$ covering $\Omega$ where the total number of grid points $N \leq N_{max}$, such that the error $e(\boldsymbol{P})$ attains a minima. For simplicity, the number of component grids in the Chimera grid as well as the background grids will be kept constant during the optimization.

# 4 Genetic algorithms

Genetic algorithms have been used in various engineering applications [8] and several books have been written on the subject [7], [6]. Here, we have used one of the methods implemented in the library GAlib[11], which we outline below.

A genetic algorithm strives to optimize an individual or group of individuals with respect to an objective function. In our application, each individual is a computational grid, and we want to optimize the grid to minimize the error when solving a Poisson equation on the grid. The optimization is performed in search space, which is the sub-domain of the parameter space that describes the permissible values of the parameters. Each grid is uniquely determined by the parameter vector $\boldsymbol{P}$, which contains all input parameters to the grid generator. Rather than moving from one point to another in search space, genetic algorithms work with an evolving population of individuals $\{\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_N\}$. A central concept in the algorithm is the fitness of an individual. Since we aim at minimizing the error $e(\boldsymbol{P})$, an individual with a small error should have a higher fitness value than one with a large error.

We therefore define the fitness of an individual $\boldsymbol{P}$ to be

$$f(\boldsymbol{P}) = -e(\boldsymbol{P}) + \max_{1 \leq i \leq N} e(\boldsymbol{P}_i) + \min_{1 \leq i \leq N} e(\boldsymbol{P}_i).$$

Hence, $f(\boldsymbol{P})$ attains its maxima where $e(\boldsymbol{P})$ is minimized, and vice versa. Furthermore, $f(\boldsymbol{P}) > 0$ since $e(\boldsymbol{P}) > 0$.

The user provides the initial individual which is used to both generate the initial population and define the search space by specifying how much each parameter is allowed to change from the value in the initial individual. In the current work, we allow each parameter to change up to 50%. The genetic algorithm starts off by initializing a population by randomly disturbing the parameters of the initial individual. The initial population constitutes the first generation and the algorithm proceeds by evaluating the objective function to obtain a fitness value for each individual. These values are used by the reproduction operator to select the best individuals for reproduction. From the fitness value, each individual is assigned a probability to be used for reproduction, where the probability for individual $q$ is taken to be the fitness fraction $f(\boldsymbol{P}_q)/\sum_{i=1}^{N} f(\boldsymbol{P}_j)$. This can be viewed as constructing a roulette wheel with variable slot sizes, where the slot size is proportional to the fitness fraction of each individual.

To select one parent, the roulette wheel selector is applied twice to pick two candidates which then are processed by the tournament selector, which picks the individual with the highest fitness value. The selection procedure is repeated to pick a second parent. With a specified probability (usually around 0.7), it is determined whether the parents should undergo crossover. If this is not the case, the first and second child are simply taken to be copies of the first and second parent, respectively. Otherwise, the parameter vectors of the parents are mixed to produce the children. In this work, a uniform crossover operator is used. Each parameter value is represented as a 32 bit floating point number, and the crossover operator "flips a coin" for each bit to decide whether the value of that bit in the first child should be taken from the first or second parent. The value of that bit in the second child is then taken from the other parent. After the children are produced, their parameter vectors undergo mutation. Some mutation is important for finding global optima, but the expectancy for mutation is kept relatively low. For each parameter in the parameter vector, it is decided according to a specified probability if that value should undergo mutation. In that case, a new value is picked according to a Gaussian distribution centered around

the current value of the parameter. The selection, crossover and mutation steps are repeated until the desired number of children have been obtained.

In the following, we will focus on the steady state genetic algorithm. In this method, a fraction of the the best individuals in the current generation are kept together with the best children to form the next generation. This fraction is called the replacement ratio. New generations evolve until a stopping criteria is met. In this work, we simply stop after a specified number of generations.

# 5 Numerical examples

In our application, each individual is represented by a command file that can be read by the grid generator. Naturally, the command file contains more information than the changeable parameters. In our implementation, the parameters that are allowed to change are indicated by a comment on the preceding line in the command file of the initial individual. It is not necessary for the grid generator to succeed in generating a valid Chimera grid for the initial individual, but the values for the parameters need to be close to those of a valid Chimera grid.

In the present work, we use the implementations of the steady state genetic algorithms provided by the library GAlib [11]. During our experiments, we will set the replacement ratio to 0.5, the expectancy for crossover to be 0.7 and vary the expectancy for mutation between 0.01 and 0.5. Furthermore, the size of the population will vary from 10 to 40 individuals and the number of generations will vary from 10 to 40.

The genetic algorithm uses a random-number generator to perform many operations and we use different seeds for each run. Note that this is the only difference between the runs. To obtain statistical information on how the parameters for the genetic algorithm should be chosen, we perform 5 runs for each parameter setting. During the grid optimization, we keep $N_{smooth} = 20$ and $I_m = 1.0$, and allow the following 9 `ogen` parameters to change: $\alpha$, $\nu$, $N_r$, $N_s$, $\epsilon$, $\gamma_s$, $\gamma_\kappa$, $\gamma_e$, and $d$. As was previously mentioned, the number of grid lines is constrained to satisfy $N_r N_s \leq N_{max}$, where $N_{max}$ is the number of grid points in the original grid.

As computational domain, we will use a wing-like profile in a rectangular box. The original grid is presented in Figure 2 and the corresponding error in the solution of the Poisson equation is shown in Figure 3. The max-norm

| Grid | Error | $\alpha$ | $\nu$ | $N_r$ | $\epsilon$ | $\gamma_s$ | $\gamma_\kappa$ | $\gamma_e$ | $d$ |
|---|---|---|---|---|---|---|---|---|---|
| Original | $2.3 \cdot 10^{-2}$ | 1.2 | 0.05 | 241 | 0.1 | 1.0 | 1.0 | 0.1 | 0.25 |
| P40-R2 | $1.74 \cdot 10^{-3}$ | 1.0052 | 0.03362 | 276 | 0.1087 | 1.282 | 0.5 | 0.05 | 0.125 |
| P20-R1 | $1.74 \cdot 10^{-3}$ | 1.0067 | 0.05960 | 287 | 0.1131 | 1.278 | 1.5 | 0.05 | 0.125 |
| P20-R4 | $1.76 \cdot 10^{-3}$ | 1.0188 | 0.04330 | 361 | 0.0597 | 1.340 | 0.77 | 0.053 | 0.125 |
| P20-R5 | $1.72 \cdot 10^{-3}$ | 1.0330 | 0.05567 | 361 | 0.05 | 1.266 | 0.5 | 0.061 | 0.125 |
| P10-R3 | $1.72 \cdot 10^{-3}$ | 0.9991 | 0.06074 | 282 | 0.05 | 0.827 | 1.39 | 0.05 | 0.125 |

Table 1: Parameter settings for the original grid and some of the best optimized grids. Grid P$xx$-R$y$ refers to population size $xx$, run $y$, in Figure 5.

of the error on the initial grid is $2.3 \times 10^{-2}$.

In our first numerical experiment, we vary the expectancy for mutation between 0.01 and 0.5. A population of 30 individuals is evolved for 10 generations in all cases. The max-norm of the errors are reported in Figure 4. Since the errors vary substantially between runs with the same mutation, it is not obvious from this comparison what the best value for the mutation is. However, it can be seen that all runs are able to reduce the error by almost an order of magnitude, compared to the error in the original grid.

In a second experiment, we fix the expectancy for mutation to 0.1 and vary the size of the population between 10 and 40, such that the number of evaluations of the objective function is 400 for all cases. Hence, the number of generations is taken to be 400 divided by the size of the population. The results are presented in Figure 5. From these runs, it seems clear that the sensitivity to the random number seed decreases as the size of the population increases. The error levels for the biggest population (40 individuals) after 10 generations are only slightly higher than the best run using half the population size (20 individuals) but twice as many generations. However, the worst run with 20 individuals is worse than the worst run with 40 individuals. This indicates that a larger population is preferable, especially if only a few different runs will be performed. The best individual for all three population sizes had an error of around $1.75 \times 10^{-3}$, and we choose to look closer at the resulting grid for run 2 with population size 40. In Figure 2, we show the optimized grid, and the corresponding error is presented in Figure 6.

Finally, we compare the parameter settings in some of the best optimized grids, see Table 1. It is interesting to see that some parameters ($\alpha$, $\gamma_e$ and

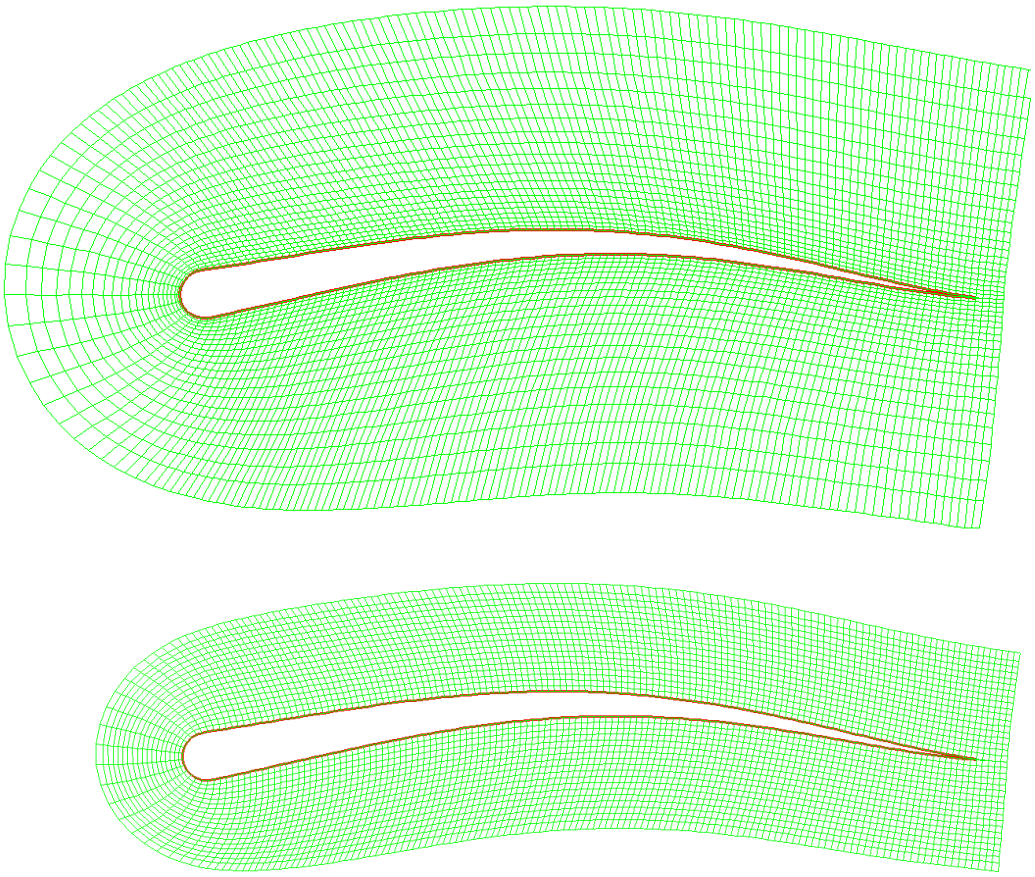

Figure 2: The body fitted component grid in the original (top) and optimized (bottom) Chimera grid. The optimized grid corresponds to run 2 with population size 40 and 10 generations.

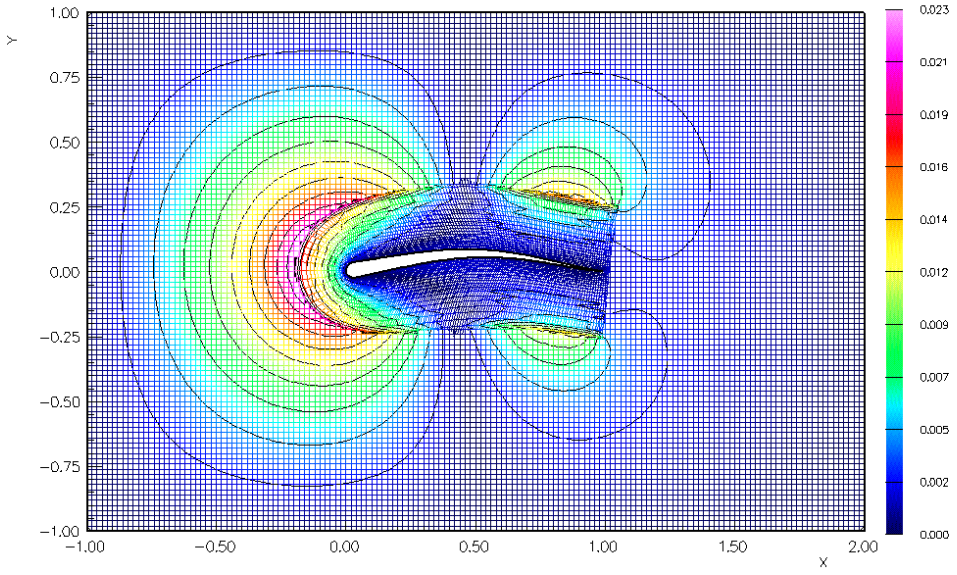


Figure 3: The error for the original Chimera grid. The maximum value is $2.3 \times 10^{-2}$.

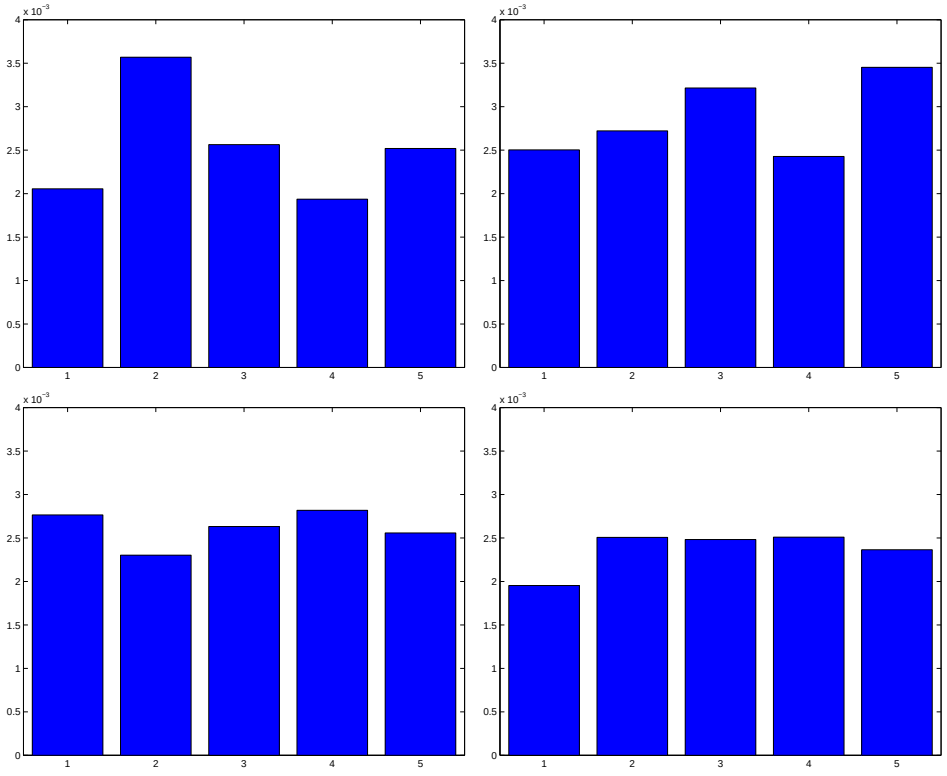


Figure 4: The error in max norm for five runs with the expectancy for mutation set to 0.01 (top left), 0.10 (top right), 0.20 (bottom left), and 0.50 (bottom right). Note that only the best individual for each run is shown.

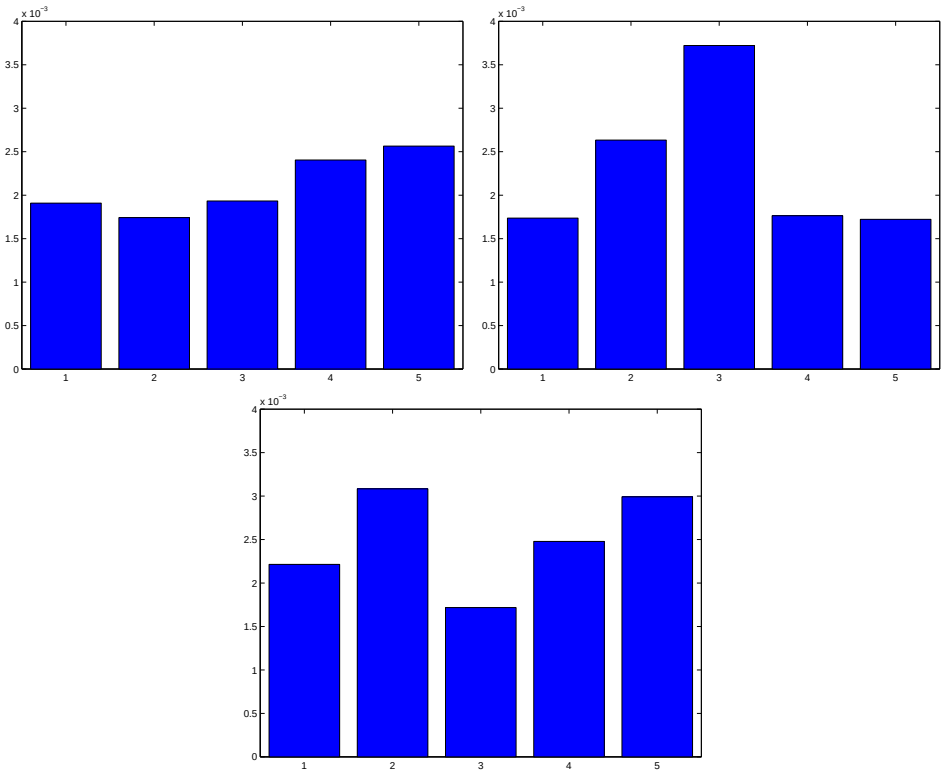


Figure 5: The error in max norm for five runs with the population size varying from 40 (top left), 20 (top right), and 10 (bottom). The number of generations was chosen to make the total number of evaluations of the objective function equal 400. Note that only the best individual for each run is shown.

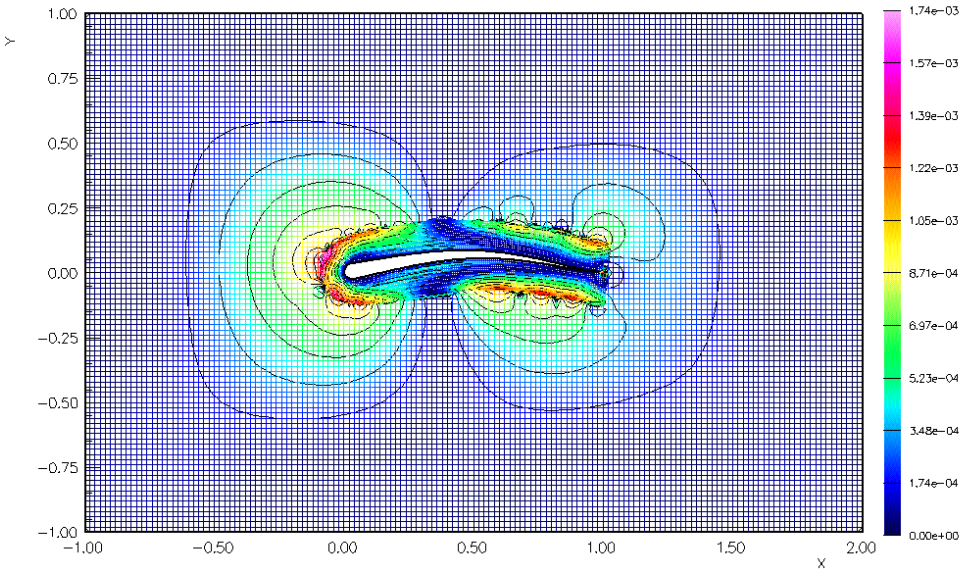


Figure 6: The error for the best Chimera grid corresponding to run 2 with population size 40 and 10 generations. The maximum value is $1.74 \times 10^{-3}$.

$d$) have very similar values in all of these grids. Other parameters ($\epsilon$ and $\gamma_\kappa$) appear to be almost redundant since they take very different values. The remaining parameters ($\nu$, $N_r$, $\gamma_s$) vary less, but still substantially enough to indicate that the parameters could be improved further. Recall that all parameters are allowed to change up to 50% from the original value. Most noticeable seems to be that all cases have the width of the grid, $d$, set to the minimum value (0.125). The same behavior is seen for the equidistribution weight, $\gamma_e$, which is close to its minimum value (0.05) for all cases. Since we are evaluating the error in Poisson's equation, it is not surprising to see that the geometric growth ratio, $\alpha$, is close to 1.0 for all runs. Note that an unit growth ratio corresponds to an unstretched grid.

# 6 Conclusions

Manual generation of grids for complex geometries can be a tedious task that requires a significant amount of time and knowledge. To efficiently use a grid generator, the user must have a thorough understanding of how all parameters in the grid generation algorithm affect the resulting grid. We have presented an optimization technique that automatically improves an initial grid by using a genetic algorithm together with the construction of exact solutions of PDE's on general domains, which enables us to use the exact error in the numerical solution as objective function. We have demonstrated that the method is capable of improving a grid such that the error in the solution of a Poisson equation is reduced by more than an order of magnitude. The technique was demonstrated using finite difference method on overlapping grid, but it should be applicable to other types of field discretization methods and other types of computational grids.

We plan to extend the technique to minimize the error for other PDE's and compare the properties of the resulting grids. In particular, it would be interesting to optimize the grid for a system of PDE's, where there are conflicting grid requirements from the different equations in the system. The incompressible Navier-Stokes equations would be an example of such a system, when there are boundary layers in the velocity, but not in the pressure. Since the computation of the error in a numerical solution of a PDE is costly, especially for three-dimensional problems, we also plan to explore parallel computing techniques.

# 7 Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

# References

[1] David L. Brown, Kyle Chand, Petri Fast, Brian Gunney, William D. Henshaw, Brian Miller, N. Anders Petersson, Bobby Philip, and Dan Quinlan. *Overture Documentation, LLNL Overlapping Grid Project*, 2000. `http://www.llnl.gov/casc/Overture`.

[2] Alex Bykat. Evolutionary triangulator. *Applied Artificial Intelligence*, 14:191–203, 2000.

[3] W. Chan and P. Buning. A hyperbolic surface grid generation scheme and its applications. *AIAA*, 94-2208, 1994.

[4] W. M. Chan and J. L. Steger. Enhancements of a three-dimensional hyperbolic grid generation scheme. *Applied Mathematics and Computation*, 51:181–205, 1992.

[5] G. Chesshire and W. D. Henshaw. Composite overlapping meshes for the solution of partial differential equations. *J. Comput. Phys.*, 90(1):1–64, 1990.

[6] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley Longman Inc., 1989.

[7] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975.

[8] K. KrishnaKumar. Genetic algorithms: an introduction and overview of their capabilities. *AIAA*, 92-4462-CP:728–738, 1992.

[9] J. A. Sethian. Curvature flow and entropy conditions applied to grid generation. *J. Comput. Phys.*, 115(2):440–454, 1994.

[10] Joe F. Thompson, Bharat K. Soni, and Nigel P. Wheatherill. *Handbook of Grid Generation.* CRC Press LLC, Boca Raton, 1998.

[11] Matthew Wall. *GAlib: A C++ Library of Genetic Algorithm Components*, 1996. `http://lancet.mit.edu/ga/`.